Vulnerability Type: Command Injection

```
const express = require('express');
const router = express.Router()

const { exec, spawn }  = require('child_process');


router.post('/ping', (req,res) => {
    exec(`${req.body.url}`, (error) => {
        if (error) {
            return res.send('error');
        }
        res.send('pong')
    })

})

router.post('/gzip', (req,res) => {
    exec(
        'gzip ' + req.query.file_path,
        function (err, data) {
          console.log('err: ', err)
          console.log('data: ', data);
          res.send('done');
    });
})

router.get('/run', (req,res) => {
   let cmd = req.params.cmd;
   runMe(cmd,res)
});

function runMe(cmd,res){
//    return spawn(cmd);

    const cmdRunning = spawn(cmd, []);
    cmdRunning.on('close', (code) => {
        res.send(`child process exited with code ${code}`);
    });
}

module.exports = router
```

Vulnerability Type: Command Injection

```
import os
from flask import Flask, request
app = Flask(__name__)
```

```
# curl -X GET "http://localhost:5000/tainted7/touch%20HELLO"
@app.route("/tainted7/<something>")
def test_sources_7(something):

    os.system(request.remote_addr)

    return "foo"

if __name__ == "__main__":
 app.run(debug=True)
```

Vulnerability Type: Denial Of Service

```
const express = require('express');
const router = express.Router()


router.post("/list-users", (req, res) => {
    var obj = req.body.users;
    var someArr = [];

    // Potential DoS if obj.length is large.
    for (var i = 0; i < obj.length; i++) {
        someArr.push(obj[i]);
    }

    //doing something with the code
    res.send(someArr.join(','));
});


module.exports = router
```

Vulnerability Type: NoSQL Injection

```
const express = require('express');
const config = require('../config')
const router = express.Router()

const MongoClient = require('mongodb').MongoClient;
const url = config.MONGODB_URI;

router.post('/customers/register', async (req, res) => {

    const client = await MongoClient.connect(url, { useNewUrlParser: true })
        .catch(err => { console.log(err); });
    if (!client) {
        return res.json({ status: "Error" });
    }
```

```
    const db = client.db(config.MONGODB_DB_NAME);
    const customers = db.collection("customers")

    let myobj = { name: req.body.name, address: req.body.address };
    customers.insertOne(myobj, function (err) {
        if (err) throw err;
        console.log("user registered");
        res.json({ status:"success", "message": "user inserted" })
        db.close();
    });

})


// Vulnerable search function
router.post('/customers/find', async (req, res) => {

    const client = await MongoClient.connect(url, { useNewUrlParser: true })
        .catch(err => { console.log(err); });
    if (!client) {
        return res.json({ status: "Error" });
    }
    const db = client.db(config.MONGODB_DB_NAME);
    const customers = db.collection("customers")

    let name = req.body.name
    let myobj = { name: name };
    customers.findOne(myobj, function (err, result) {
        if (err) throw err;
        db.close();
        res.json(result)
    });


})

// Vulnerable Authentication
// Authentication Bypass Example
//   curl   -X   POST   http://localhost:3000/customers/login/   --data   "{\"email\":
{\"\$gt\":\"\"} , \"password\": {\"\$gt\":\"\"}}" -H "Content-Type: application/json"

router.post('/customers/login', async (req, res) => {

    const client = await MongoClient.connect(url, { useNewUrlParser: true })
        .catch(err => { console.log(err); });
    if (!client) {
        return res.json({ status: "Error" });
    }
    const db = client.db(config.MONGODB_DB_NAME);
    const customers = db.collection("customers")

    let myobj = { email: req.body.email, password: req.body.password };
    customers.findOne(myobj, function (err, result) {
```

```
        if (err) throw err;
        db.close();
        res.json(result)
    });


})

module.exports = router
```

Vulnerability Type: Open Redirect

```
const Koa = require('koa');
const urlLib = require('url');
const app = new Koa();

app.use(async ctx => {
 var url = ctx.query.target;
 ctx.redirect(url);
});

app.listen(3000);
```

Vulnerability Type: Open Redirect

```
console.log('WIP')
const express = require('express');
const router = express.Router()

router.get('/login',function(req, res){
    let followPath = req.query.path;
    if(req.session.isAuthenticated()){
        res.redirect('http://example.com/'+followPath); //false positive
    }else{
        res.redirect('/');
    }
});

router.get('/goto',function(req, res){
    let url = encodeURI(req.query.url); //vulnerability
    res.redirect(url);
});


module.exports = router
```

Vulnerability Type: Path Traversal

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  const file = readFile(req.query.name).toString()
  res.send(file)
})


function readFile(path){

    result = fs.readFileSync(path)
    return result;

  }



app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`)
})


Vulnerability Type: Path Traversal


const { ApolloServer, gql } = require('apollo-server');

var fs = require('fs');
var express = require('express');
var app = express();


// A schema is a collection of type definitions (hence "typeDefs")
// that together define the "shape" of queries that are executed against
// your data.
const typeDefs = gql`
  # Comments in GraphQL strings (such as this one) start with the hash (#) symbol.

  # This "Book" type defines the queryable fields for every book in our data source.
  type Book {
    title: String
    author: String
  }

  # The "Query" type is special: it lists all of the available queries that
  # clients can execute, along with the return type for each. In this
  # case, the "books" query returns an array of zero or more Books (defined above).
  type Query {
    books(path: String): [Book]
```

```
  }
`;


const books = [
    {
      title: 'The Awakening',
      author: 'Kate Chopin',
    },
    {
      title: 'City of Glass',
      author: 'Paul Auster',
    },
  ];



  // Resolvers define the technique for fetching the types defined in the
// schema. This resolver retrieves books from the "books" array above.
const resolvers = {
    Query: {
      books: (parent, args, context, info) => {
        const file = readFile(args.path).toString()
        console.log(file)
        return [{title: file, author: "hello"}]
    },
    },
  };

function readFile(path ){

  result = fs.readFileSync(path)
  return result;

}

  // The ApolloServer constructor requires two parameters: your schema
// definition and your set of resolvers.
const server = new ApolloServer({
    introspection: true,
    typeDefs, resolvers });



// The `listen` method launches a web server.
server.listen().then(({ url }) => {
  console.log(`  Server ready at ${url}`);
});
```

Vulnerability Type: Path Traversal

```
import os

from flask import (
    Flask,
    render_template,
    request,
    url_for,
    redirect,
    session,
    render_template_string
)
from flask.ext.session import Session

app = Flask(__name__)


execfile('flag.py')
execfile('key.py')

FLAG = flag
app.secret_key = key


@app.route("/golem", methods=["GET", "POST"])
def golem():
    if request.method != "POST":
        return redirect(url_for("index"))

    golem = request.form.get("golem") or None

    if golem is not None:
        golem = golem.replace(".", "").replace(
            "_", "").replace("{", "").replace("}", "")

    if "golem" not in session or session['golem'] is None:
        session['golem'] = golem

    template = None

    if session['golem'] is not None:
        template = '''{% % extends "layout.html" % %}
  {% % block body % %}
  <h1 > Golem Name < /h1 >
  <div class ="row >
  <div class = "col-md-6 col-md-offset-3 center" >
  Hello: % s, why you don't look at our <a href=' / article?name = article'> article <
/a >?
  < / div >
  < / div >
  {% % endblock % %}
  ''' % session['golem']
```

```
        print

        session['golem'] = None

    return render_template_string(template)


@app.route("/", methods=["GET"])
def index():
    return render_template("main.html")


@app.route('/article', methods=['GET'])
def article():

    error = 0

    if 'name' in request.args:
        page = request.args.get('name')
    else:
        page = 'article'

    if page.find('flag') >= 0:
        page = 'notallowed.txt'

    try:
        template = open('/home/golem/articles/{}'.format(page)).read()
    except Exception as e:
        template = e

    return render_template('article.html', template=template)

if __name__ == "__main__":
    app.run(host='0.0.0.0', debug=False)
```

Vulnerability Type: PostMessage Security

```
//https://html5.digi.ninja/challenge.html

if (typeof(SERVER_DOMAIN) === 'undefined') {
 window.location.replace("/unconfigured.html");
}

const RECEIVE_URL = SERVER_DOMAIN + "/challenge_scoreboard.html" + "?origin=" +
get_domain();

var window_ref = null;

document.getElementById("username").focus();
```

```
function store_username() {
 var username;
 var username_obj;

 username_obj = document.getElementById("username");
 username = username_obj.value

 var welcome;
 welcome = document.getElementById("welcome");
 welcome.innerHTML = "Welcome " + html_encode (username);

 var set_username;
 set_username = document.getElementById("set_username");
 set_username.style.display="none";

 var game;
 game = document.getElementById("game");
 game.style.display="inline";

 start_game();
 // have to do time out so the window can open
 setTimeout (function () {send_username(username);}, 1000);

 return false;
}

function check_guess() {
 var guess_obj = document.getElementById("guess");
 var guess = guess_obj.value;
 var res = document.getElementById("result");

 send_message("guess:" + guess);

 document.getElementById("guess").focus();
 document.getElementById("guess").value = "";
}

function html_encode (html) {
 return document.createElement( 'a' ).appendChild(
  document.createTextNode( html ) ).parentNode.innerHTML;
}

function send_message(message) {
 if (window_ref == null) {
  return;
 }
 if (window_ref.closed) {
  return;
 }

 window_ref.postMessage(message, "*");
 // window_ref.postMessage(message, RECEIVE_URL);
}
```

```
function start_game() {
 open_window();
 document.getElementById("guess").focus();
}

function send_username(username) {
 message = "user:" + html_encode(username);
 send_message(message);
}

function get_domain() {
 var url = window.location.href
 var arr = url.split("/");
 return arr[0] + "//" + arr[2]
}

function open_window() {
 if (window_ref == null || window_ref.closed) {
  window_ref = window.open (RECEIVE_URL, "score board", "height=260,width=550");

  if (window_ref == null) {
   alert ("Failed to open window. You must allow pop-ups.");
  }
 }
}

const usernameButton = document.getElementById("setUsername");
usernameButton.addEventListener("click", store_username, false);

const guessButton = document.getElementById("checkGuess");
guessButton.addEventListener("click", check_guess, false);

start_game();


Vulnerability Type: PostMessage Security


//https://html5.digi.ninja

if (typeof(SERVER_DOMAIN) === 'undefined') {
 window.location.replace("/unconfigured.html");
}

const RECEIVE_URL = SERVER_DOMAIN + "/s_child.html" + "?origin=" + get_domain();

var window_ref = null;

function send_message(destination) {
 message = document.getElementById("message").value;
 receiver.contentWindow.postMessage(message, SERVER_DOMAIN);
}
```

```
function get_domain() {
 var url = window.location.href
 var arr = url.split("/");
 return arr[0] + "//" + arr[2]
}

var receiver = document.getElementById("s_iframe");
receiver.src = RECEIVE_URL;

const sendMessageButton = document.getElementById("send_message_button");
sendMessageButton.addEventListener("click", send_message, false);
```

Vulnerability Type: PostMessage Security

```
function receiveMessage(message) {
 let tokenSpan = document.getElementById("token");
 if (message.data == null) {
  tokenSpan.innerText = "<Unset>";
 } else {
  tokenSpan.innerText = message.data;
 }
}

window.addEventListener("message", receiveMessage, false);
```

Vulnerability Type: Prototype Pollution

```
const express = require('express');
const router = express.Router()

const lodash = require('lodash');

//if  req.body.config  ==  '{"constructor":  {"prototype":  {"isAdmin":  true}}}'  it  will
bypass the authentication
function check(req, res) {

    let config = {};
    lodash.defaultsDeep(config, JSON.parse(req.body.config));

    let user = getCurrentUser();
    if(!user){
      user = {};
    }

    if (user.isAdmin && user.isAdmin === true) {
        res.send('Welcome Admin')
    }else{
        res.send('Welcome User')
```

```
    }
}

//fake function that get current user from session or db
function getCurrentUser(){
  return false;
}


router.post('/check-user',check)

module.exports = router
```

Vulnerability Type: ReDoS

```
const express = require('express');
const router = express.Router()


router.get("/tstMe", (req, res) => {
    var r = /([a-z]+)+$/;

    let match = r.test(req.params.id);
    res.send(match)

});


module.exports = router
```

Vulnerability Type: SQL Injection

```
var mysql = require('db-mysql');
var http = require('http');
var out;
var valTom;
var req = http.request(options, function(res)
{
 res.on('data', function(chunk)
 {
  valTom = chunk;
 }
 );
}
);
new mysql.Database(
{
 hostname: 'localhost',
 user: 'user',
```

```
 password: 'password',
 database: 'test'
}
).connect(function(error)
{
 var the_Query =
 "INSERT INTO Customers (CustomerName, ContactName) VALUES ('Tom'," +
 valTom + ")";
 this.query(the_Query).execute(function(error, result)
 {
  if (error)
  {
   console.log("Error: " + error);
  }
  else
  {
   console.log('GENERATED id: ' + result.id);
  }
 }
 );
 out = resIn;
}
);
```

Vulnerability Type: SQL Injection

```
const express = require('express');
const router = express.Router()

const config = require('../../config')
const mysql      = require('mysql');
const connection = mysql.createConnection({
  host     : config.MYSQL_HOST,
  port     : config.MYSQL_PORT,
  user     : config.MYSQL_USER,
  password : config.MYSQL_PASSWORD,
  database : config.MYSQL_DB_NAME,
});

connection.connect();

router.get('/example1/user/:id', (req,res) => {
    let userId = req.params.id;
    let query = {
        sql : "SELECT * FROM users WHERE id=" + userId
    }
    connection.query(query,(err, result) => {
        res.json(result);
    });
})
```

```
router.get('/example2/user/:id',  (req,res) => {
    let userId = req.params.id;
    connection.query("SELECT * FROM users WHERE id=" + userId,(err, result) => {
        res.json(result);
    });
})

router.get('/example3/user/:id',  (req,res) => {
    let userId = req.params.id;
    connection.query({
        sql : "SELECT * FROM users WHERE id=" +userId
    },(err, result) => {
        res.json(result);
    });
})


module.exports = router
```

Vulnerability Type: SQL Injection

```
var express = require('express')

var app = express()
const Sequelize = require('sequelize');
const sequelize = new Sequelize('database', 'username', 'password', {
  dialect: 'sqlite',
  storage: 'data/juiceshop.sqlite'
});

app.post('/login', function (req, res) {
    sequelize.query('SELECT * FROM Products WHERE name LIKE ' +  req.body.username);
  })
```

Vulnerability Type: SSRF

```
const express = require('express');
const router = express.Router()
const request = require('request');

router.post('/downlad-url', (req, res) => {
    downloadURL(req.body.url, () =>{
        res.send('Done')
    })
});

const downloadURL = (url, onend) => {
    const opts = {
      uri: url,
      method: 'GET',
```

```
      followAllRedirects: true
    }

    request(opts)
      .on('data', ()=>{})
      .on('end', () => onend())
      .on('error', (err) => console.log(err, 'controller.url.download.error'))
}

module.exports = router
```

Vulnerability Type: Server Side Template Injection

```
import os

from flask import (
    Flask,
    render_template,
    request,
    url_for,
    redirect,
    session,
    render_template_string
)
from flask.ext.session import Session

app = Flask(__name__)


execfile('flag.py')
execfile('key.py')

FLAG = flag
app.secret_key = key


@app.route("/golem", methods=["GET", "POST"])
def golem():
    if request.method != "POST":
        return redirect(url_for("index"))

    golem = request.form.get("golem") or None

    if golem is not None:
        golem = golem.replace(".", "").replace(
            "_", "").replace("{", "").replace("}", "")

    if "golem" not in session or session['golem'] is None:
        session['golem'] = golem

    template = None
```

```
    if session['golem'] is not None:
        template = '''{% % extends "layout.html" % %}
  {% % block body % %}
  <h1 > Golem Name < /h1 >
  <div class ="row >
  <div class = "col-md-6 col-md-offset-3 center" >
  Hello: % s, why you don't look at our <a href=' / article?name = article'> article <
/a >?
  < / div >
  < / div >
  {% % endblock % %}
  ''' % session['golem']

        print

        session['golem'] = None

    return render_template_string(template)


@app.route("/", methods=["GET"])
def index():
    return render_template("main.html")


@app.route('/article', methods=['GET'])
def article():

    error = 0

    if 'name' in request.args:
        page = request.args.get('name')
    else:
        page = 'article'

    if page.find('flag') >= 0:
        page = 'notallowed.txt'

    try:
        template = open('/home/golem/articles/{}'.format(page)).read()
    except Exception as e:
        template = e

    return render_template('article.html', template=template)

if __name__ == "__main__":
    app.run(host='0.0.0.0', debug=False)
```

Vulnerability Type: Server Side Template Injection

```
from jinja2 import Template
from flask import request

import flask

app = flask.Flask(__name__)
app.config['DEBUG'] = True

@app.route('/', methods=['GET'])
def home():
    renderer = Template('Hello, ' + request.args['name'])
    return renderer.render()

app.run()
```

Vulnerability Type: Unsafe Deserialization

```
class Vault(object):
    '''R/W an ansible-vault yaml file'''

    def __init__(self, password):
        self.password = password
        self.vault = VaultLib(password)

    def load(self, stream):
        '''read vault steam and return python object'''
        return yaml.load(self.vault.decrypt(stream)) [0]
```

Vulnerability Type: Unsafe Deserialization

```
//safeLoadAll and jsyaml.safeLoad are vulnerable if DEFAULT_FULL_SCHEMA is used
const jsyaml = require("js-yaml");

var express = require('express');
var app = express();
app.post('/store/:id', function(req, res) {
  let data;
  let unsafeConfig = { schema: jsyaml.DEFAULT_FULL_SCHEMA };
  data = jsyaml.safeLoad(req.params.data, unsafeConfig);
```

Vulnerability Type: Unsafe Deserialization

```
# Python's revenge
# This is a easy python sandbox, can you bypass it and get the flag?
# https://hitbxctf2018.xctf.org.cn/contest_challenge/
from __future__ import unicode_literals
```

```
from flask import Flask, request, make_response, redirect, url_for, session
from flask import render_template, flash, redirect, url_for, request
from werkzeug.security import safe_str_cmp
from base64 import b64decode as b64d
from base64 import b64encode as b64e
from hashlib import sha256
from cStringIO import StringIO
import random
import string

import os
import sys
import subprocess
import commands
import pickle
import cPickle
import marshal
import os.path
import filecmp
import glob
import linecache
import shutil
import dircache
import io
import timeit
import popen2
import code
import codeop
import pty
import posixfile

SECRET_KEY = 'you will never guess'

if not os.path.exists('.secret'):
    with open(".secret", "w") as f:
        secret = ''.join(random.choice(string.ascii_letters + string.digits)
                         for x in range(4))
        f.write(secret)
with open(".secret", "r") as f:
    cookie_secret = f.read().strip()

app = Flask(__name__)
app.config.from_object(__name__)

black_type_list = [eval, execfile, compile, open, file, os.system, os.popen, os.popen2,
os.popen3, os.popen4, os.fdopen, os.tmpfile, os.fchmod, os.fchown, os.open, os.openpty,
os.read, os.pipe, os.chdir, os.fchdir, os.chroot, os.chmod, os.chown, os.link,
os.lchown, os.listdir, os.lstat, os.mkfifo, os.mknod, os.access, os.mkdir, os.makedirs,
os.readlink, os.remove, os.removedirs, os.rename, os.renames, os.rmdir, os.tempnam,
os.tmpnam, os.unlink, os.walk, os.execl, os.execle, os.execlp, os.execv, os.execve,
os.dup, os.dup2, os.execvp, os.execvpe, os.fork, os.forkpty, os.kill, os.spawnl,
os.spawnle, os.spawnlp, os.spawnlpe, os.spawnv, os.spawnve, os.spawnvp, os.spawnvpe,
pickle.load, pickle.loads, cPickle.load, cPickle.loads, subprocess.call,
```

```
subprocess.check_call, subprocess.check_output, subprocess.Popen, commands.getstatusoutput, commands.getoutput, commands.getstatus, glob.glob, linecache.getline, shutil.copyfileobj, shutil.copyfile, shutil.copy, shutil.copy2, shutil.move, shutil.make_archive, dircache.listdir, dircache.opendir, io.open, popen2.popen2, popen2.popen3, popen2.popen4, timeit.timeit, timeit.repeat, sys.call_tracing, code.interact, code.compile_command, codeop.compile_command, pty.spawn, posixfile.open, posixfile.fileopen]


@app.before_request
def count():
    session['cnt'] = 0


@app.route('/')
def home():
      remembered_str = 'Hello, here\'s what we remember for you. And you can change, delete or extend it.'
     new_str = 'Hello fellow zombie, have you found a tasty brain and want to remember where? Go right here and enter it:'
    location = getlocation()
    if location == False:
        return redirect(url_for("clear"))
    return render_template('index.html', txt=remembered_str, location=location)


@app.route('/clear')
def clear():
    flash("Reminder cleared!")
    response = redirect(url_for('home'))
    response.set_cookie('location', max_age=0)
    return response


@app.route('/reminder', methods=['POST', 'GET'])
def reminder():
    if request.method == 'POST':
        location = request.form["reminder"]
        if location == '':
            flash("Message cleared, tell us when you have found more brains.")
        else:
            flash("We will remember where you find your brains.")
        location = b64e(pickle.dumps(location))
        cookie = make_cookie(location, cookie_secret)
        response = redirect(url_for('home'))
        response.set_cookie('location', cookie)
        return response
    location = getlocation()
    if location == False:
        return redirect(url_for("clear"))
    return render_template('reminder.html')
```

```
class FilterException(Exception):
    def __init__(self, value):
        super(FilterException, self).__init__(
            'The callable object {value} is not allowed'.format(value=str(value)))


class TimesException(Exception):
    def __init__(self):
        super(TimesException, self).__init__(
            'Call func too many times!')


def _hook_call(func):
    def wrapper(*args, **kwargs):
        session['cnt'] += 1
        print session['cnt']
        print args[0].stack
        for i in args[0].stack:
            if i in black_type_list:
                raise FilterException(args[0].stack[-2])
            if session['cnt'] > 4:
                raise TimesException()
        return func(*args, **kwargs)
    return wrapper


def loads(strs):
    reload(pickle)
    files = StringIO(strs)
    unpkler = pickle.Unpickler(files)
    unpkler.dispatch[pickle.REDUCE] = _hook_call(
        unpkler.dispatch[pickle.REDUCE])
    return unpkler.load()


def getlocation():
    cookie = request.cookies.get('location')
    if not cookie:
        return ''
    (digest, location) = cookie.split("!")
    if not safe_str_cmp(calc_digest(location, cookie_secret), digest):
        flash("Hey! This is not a valid cookie! Leave me alone.")
        return False
    location = loads(b64d(location))
    return location


def make_cookie(location, secret):
    return "%s!%s" % (calc_digest(location, secret), location)


def calc_digest(location, secret):
    return sha256("%s%s" % (location, secret)).hexdigest()
```

```
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=5051)
```

Vulnerability Type: Unsafe Deserialization

```
var express = require('express');
var cookieParser = require('cookie-parser');
var escape = require('escape-html');
var serialize = require('node-serialize');
var app = express();
app.use(cookieParser())

app.get('/', function(req, res) {
 if (req.cookies.profile) {
   var str = new Buffer(req.cookies.profile, 'base64').toString();
   var obj = serialize.unserialize(str);
   if (obj.username) {
     res.send("Hello " + escape(obj.username));
   }
 } else {
                                                                    res.cookie('profile',
"eyJ1c2VybmFtZSI6ImFqaW4iLCJjb3VudHJ5IjoiaW5kaWEiLCJjaXR5IjoiYmFuZ2Fsb3JlIn0=", {
       maxAge: 900000,
       httpOnly: true
     });
 }
 res.send("Hello World");
});
app.listen(3000);
```

Vulnerability Type: XSS

```
<script>
window.addEventListener('message', writeMessage, false);
function writeMessage(event)
{
 document.getElementById("message").innerHTML = event.data;
}
</script>
```

Vulnerability Type: XSS

```
const express = require('express')
const router = express.Router()
```

```
router.get('/greeting', (req, res) => {
    const { name }  = req.query;
    res.send('<h1> Hello :'+ name +"</h1>")
})

router.get('/greet-template', (req,res) => {
    name = req.query.name
    res.render('index', { user_name: name});
})

module.exports = router
```

Vulnerability Type: XXE

```
const express = require('express')
const libxmljs = require('libxml')
const db = require('db');
const router = express.Router()

router.post('/upload-products', (req, res) => {
    const XMLfile = req.files.products.data;
 const products = libxmljs.parseXmlString(XMLfile, {noent:true,noblanks:true})

 products.root().childNodes().forEach(product => {
  let newProduct = new db.Product()
  newProduct.name = product.childNodes()[0].text()
  newProduct.description = product.childNodes()[3].text()
  newProduct.save()
    });

    res.send('Thanks')
})

module.exports = router
```